CONTENTS

5 Modelling and Model Fitting

5.1 Matrix and Vector Operations

5.1.1 Vector

Definition 1. A column of real numbers is called a **vector**.

Example 1.

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} 1 \\ -3 \\ 2 \end{bmatrix} \quad \mathbf{1} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

```
Python-code:
import numpy as np
a = np.array([[1],[-3],[2]])
print("a=",a)
One5 = np.ones(5)
print("One5=",One5)
Output:
a= [[ 1]
 [-3]
 [ 2]]
One5= [1. 1. 1. 1. 1.]
```

Since $\mathbf{y}$ has n elements it is said to have **order** (or dimension) n.

5.1.2 Matrix

Definition 2.
A rectangular array of elements with m rows and k columns is called an $m \times k$ **matrix.**

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix}$$

This matrix is said to be of **order** (or dimension) $m \times k$, where

- m is the **row** order (dimension)
- k is the **column** order (dimension)
- a_{ij} is the (i, j) element of $\mathbf{A}$.

Example 2.

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & -2 \\ 0 & 4 & 5 \end{bmatrix} \quad \mathbf{I_3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{J_3} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

```
Python-code:
import numpy as np
A = np.array([[1, 3, -2], [0, 4, 5]])
I = np.eye(3)
One = np.ones(3)
J = np.ones((3,3))
print("A=", A)
print("I3=",I)
print("13=", One)
print("J3=",J)
output:
A= [[ 1  3 -2]
 [ 0  4  5]]
I3= [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
13= [1. 1. 1.]
J3= [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

5.1.3 Matrix Addition and Subtraction

Definition 3. Matrix addition
If $\mathbf{A}$ and $\mathbf{B}$ are both $m \times k$ matrices, then

$$\begin{aligned}\mathbf{C} &= \mathbf{A} + \mathbf{B} \\ &= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1k} \\ b_{21} & b_{22} & \cdots & b_{2k} \\ \vdots & \vdots & & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mk} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1k}+b_{1k} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2k}+b_{2k} \\ \vdots & \vdots & & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mk}+b_{mk} \end{bmatrix}\end{aligned}$$

Notation:

$$C_{m\times k} = \{c_{ij}\} \text{ where } c_{ij} = a_{ij} + b_{ij}$$

Definition 4. Matrix subtraction

If $\mathbf{A}$ and $\mathbf{B}$ are $m \times k$ matrices, then $\mathbf{C} = \mathbf{A} - \mathbf{B}$ is defined by

$$\mathbf{C} = \{c_{ij}\} \text{ where } c_{ij} = a_{ij} - b_{ij} \ .$$

Example 3.

$$\begin{bmatrix} 3 & 6 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 7 & -4 \\ -3 & 2 \end{bmatrix} = \begin{bmatrix} 10 & 2 \\ -1 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & -1 \\ 2 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ -1 & 1 \\ 0 & -1 \end{bmatrix}$$

```
Python-code:
import numpy as np
A1 = np.array([[3, 6], [2,1]])
A2= np.array([[7, -4], [-3, 2]])
A3 = A1 + A2
print("A1+A2=",A3)

output:
A1+A2= [[10  2]
 [-1  3]]
import numpy as np
B1 = np.array([[1, -1], [1,1],[1,0]])
B2= np.array([[1, -1], [2, 0],[1,1]])
B3 = B1 - B2
print("B1-B2=",B3)

Output:
B1-B2= [[ 0  0]
 [-1  1]
 [ 0 -1]]
```

5.1.4 Scalar Multiplication

Definition 5. Scalar multiplication

Let a be a scalar and $\mathbf{B} = \{b_{ij}\}$ be an $m \times k$ matrix, then

$$a\,\mathbf{B} = \mathbf{B}\,a = \{a\,b_{ij}\}$$

Example 4.

$$2\begin{bmatrix} 2 & -1 & 3 \\ 0 & 4 & -2 \end{bmatrix} = \begin{bmatrix} 4 & -2 & 6 \\ 0 & 8 & -4 \end{bmatrix}$$

```
 Python-Code:
import numpy as np
B1 = np.array([[2, -1, 3], [0,4,-2]])
B2 = 2*B1
print("2*B1=",B2)
Output:
B2=2*B1= [[ 4 -2  6]
          [ 0  8 -4]]
```

5.1.5 Transpose

Definition 6. Transpose
The transpose of the $m \times k$ matrix $\mathbf{A} = \{a_{ij}\}$ is the $k \times m$ matrix with elements $\{a_{ji}\}$. The transpose of $\mathbf{A}$ is denoted by $\mathbf{A^T}$ (or $\mathbf{A'}$).

Example 5.

$$\mathbf{A} = \begin{bmatrix} 1 & 4 \\ 3 & 0 \\ -2 & 6 \end{bmatrix} \qquad \mathbf{A^T} = \begin{bmatrix} 1 & 3 & -2 \\ 4 & 0 & 6 \end{bmatrix}$$

```
R-code:
import numpy as np
B1 = np.array([[1,4], [3,0],[-2,6]])
B2 = B1.T
print("B1^T = ",B2)

Output:
B1^T =
[[ 1  3 -2]
 [ 4  0  6]]
```

5.1.6 Inner Product

Definition 7. Inner product (crossproduct) of two vectors of order n

$$\mathbf{a}^T\mathbf{y} = [a_1, a_2, \cdots a_n] \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = a_1y_1 + a_2y_2 + \cdots + a_ny_n$$

Example 6.

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 11 \\ 15 \\ 23 \end{bmatrix}$$

$$\mathbf{a^T y} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 11 \\ 15 \\ 23 \end{bmatrix} = 1(11) + 2(15) + 3(23) = 110$$

```
Python-codes:
import numpy as np
a = np.array([[1, 2, 3]])
y = np.array([[11, 15, 23]])
aty = np.inner(a,y)
print("a^Ty=",aty)
Output:
a^Ty= [[110]]
```

5.1.7 Matrix Multiplication

Definition 8. Matrix multiplication
The product of an $n \times k$ matrix $\mathbf{A}$ and a $k \times m$ matrix $\mathbf{B}$ is the $n \times m$ matrix $\mathbf{C} = \{c_{ij}\}$ with elements

$$c_{ij} = a_{i1}\, b_{1j} + a_{i2}\, b_{2j} + \cdots + a_{ik}\, b_{kj}$$

Example 7.

$$\mathbf{A} = \begin{bmatrix} 3 & 0 & -2 \\ 1 & -1 & 4 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} \quad \mathbf{C} = \mathbf{AB} = \begin{bmatrix} 1 & -3 \\ 4 & 11 \end{bmatrix}$$

```
Python-codes:
import numpy as np
A = np.array([[3, 0, -2], [1, -1, 4]])
B = np.array([[1,1],[1, 2],[1,3]])
C = A@B
print("C=AB",C)
Output:
C=AB=
[[ 1 -3]
 [ 4 11]]
```

Example 8.

Consider the following matrix:

$$\mathbf{A} = \begin{bmatrix} 16.4 & 15.4 & 14.4 \\ 12.4 & 11.4 & 13.4 \\ 12.1 & 15.6 & 10.1 \end{bmatrix}.$$

Use Python to find the $(3, 2)$ element of $\mathbf{A^T A}$.

Sol:
$A^T A$ = [[569.13 582.68 524.53] [582.68 610.48 532.08] [524.53 532.08 488.93]]
$(3, 2)^{th}$ element of $\mathbf{A^T A}$ = 532.0800000000002

```
import numpy as np
A = np.array([[16.4,15.4,14.4],[12.4,11.4,13.4],
              [12.1,15.6,10.1]])
ATA = A.T@A
print("A^TA=",ATA)
ije = ATA[2,2]
print("(2,2) element of A = ", ije)
Output:
A^TA= [[606.11 604.14 558.61]
 [604.14 609.46 553.04]
 [558.61 553.04 520.11]]
(3,3) element of A =  520.1099999999999
```

5.1.8 Elementwise multiplication

Definition 9. Elementwise multiplication of two matrices

$$\mathbf{A} \# \mathbf{B} = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & & \vdots \\ a_{k1} & \cdots & a_{km} \end{bmatrix} \# \begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & & \vdots \\ b_{k1} & \cdots & b_{km} \end{bmatrix} = \begin{bmatrix} a_{11}\,b_{11} & \cdots & a_{1m}\,b_{1m} \\ \vdots & & \vdots \\ a_{k1}\,b_{k1} & \cdots & a_{km}\,b_{km} \end{bmatrix}$$

Example 9.

$$\begin{bmatrix} 3 & 1 \\ 2 & 4 \\ 0 & 6 \end{bmatrix} \# \begin{bmatrix} 1 & -5 \\ -3 & 4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 3 & -5 \\ -6 & 16 \\ 0 & 12 \end{bmatrix}$$

```
Python-codes:
import numpy as np
A = np.array([[3, 1], [2, 4],[0,6]])
B = np.array([[1,-5],[-3, 4],[-2,2]])
C = A*B
print("C=AB=",C)
Output:
C=AB=
[[ 3 -5]
 [-6 16]
 [ 0 12]]
```

5.1.9 Kronecker Product

Definition 10. Kronecker product of two matrices

$$\mathbf{A}_{k\times m} \otimes \mathbf{B}_{n\times s} = \begin{bmatrix} a_{11}\,\mathbf{B} & a_{12}\,\mathbf{B} & \cdots & a_{1m}\,\mathbf{B} \\ a_{21}\,\mathbf{B} & a_{22}\,\mathbf{B} & \cdots & a_{2m}\,\mathbf{B} \\ \vdots & \vdots & & \vdots \\ a_{k1}\,\mathbf{B} & a_{k2}\,\mathbf{B} & \cdots & a_{km}\,\mathbf{B} \end{bmatrix}$$

Example 10.

$$\begin{bmatrix} 2 & 4 \\ 0 & -2 \\ 3 & -1 \end{bmatrix} \otimes \begin{bmatrix} 5 & 3 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 6 & 20 & 12 \\ 4 & 2 & 8 & 4 \\ 0 & 0 & -10 & -6 \\ 0 & 0 & -4 & -2 \\ 15 & 9 & -5 & -3 \\ 6 & 3 & -2 & -1 \end{bmatrix}$$

$$\mathbf{a} \otimes \mathbf{y} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \otimes \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_1\, y_1 \\ a_1\, y_2 \\ a_2\, y_1 \\ a_2\, y_2 \\ a_3\, y_1 \\ a_3\, y_2 \end{bmatrix}$$

```
import numpy as np
A = np.array([[2, 4], [0, -2],[3,-1]])
B = np.array([[5,3],[2, 1]])
C = np.kron(A,B)
print("C=AB =", C)
print(C)
Output:
C=AB =
[[ 10   6  20  12]
 [  4   2   8   4]
 [  0   0 -10  -6]
 [  0   0  -4  -2]
 [ 15   9  -5  -3]
```

5.1.10 Determinant

Definition 11. The **determinant** of an $n \times n$ matrix $\mathbf{A}$ is

$$|\mathbf{A}| = \sum_{j=1}^{n} a_{ij}(-1)^{i+j}|M_{ij}| \quad \text{for any row } i$$

or

$$|\mathbf{A}| = \sum_{i=1}^{n} a_{ij}(-1)^{i+j}|M_{ij}| \quad \text{for any column } j$$

where M_{ij} is the "minor" for a_{ij} obtained by deleting the i^{th} row and j^{th} column from A.

Example 11.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$|\mathbf{A}| = a_{11}(-1)^{1+1}|a_{22}| + a_{12}(-1)^{1+2}|a_{21}|$$

then $\begin{vmatrix} 7 & 2 \\ 4 & 5 \end{vmatrix} =$

Example 12.

$$|\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

$$= a_{11}(-1)^{1+1} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} + a_{12}(-1)^{1+2} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix}$$

$$+ a_{13}(-1)^{1+3} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

$$\text{then } \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} =$$

```
Python Code:
import numpy as np
A = np.array([[7, 2], [4, 5]])
B = np.array([[1,2,3],[4, 5,6],[7,8,9]])
DA = np.linalg.det(A)
DB = np.linalg.det(B)
print("DA=|A|=", DA)
print("DB=|B|=",DB)
Output:
DA=|A|=  27.0
DB=|B|=  6.66133814775094e-16
```

5.1.11 Inverse

Definition 12. The **identity matrix**, denoted by $\mathbf{I}$, is a $k \times k$ matrix of the form

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

Example 13. $\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

```
I4 = np.eye(4)
print(''I4='',I4)
```

Definition 13. The **inverse** of a square, non-singular matrix $\mathbf{A}$ is the matrix, denoted by $\mathbf{A}^{-1}$, such that

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = I$$

Example 14.

Determine the inverse of

$$\mathbf{A} = \begin{pmatrix} 7 & 2 \\ 4 & 5 \end{pmatrix} = \text{and } \mathbf{B} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
import numpy as np
from scipy import linalg
A = np.array([[7, 2], [4, 5]])
B = np.array([[1,2,3],[4, 5,6],[7,8,9]])
IA = linalg.inv(A)
IB = linalg.inv(B)
print("IA=A^(-1)=", IA)
print("IB=B^(-1)=",IB)
Output:
IA=A^(-1)= [[ 0.18518519 -0.07407407]
 [-0.14814815  0.25925926]]
IB=B^(-1)= [[-4.50359963e+15  9.00719925e+15 -4.50359963e+15]
 [ 9.00719925e+15 -1.80143985e+16  9.00719925e+15]
 [-4.50359963e+15  9.00719925e+15 -4.50359963e+15]]
```

5.1.12 Trace

Definition 14. The **trace** of a $k \times k$ matrix $\mathbf{A} = \{a_{ij}\}$ is the sum of the diagonal elements:

$$tr(\mathbf{A}) = \sum_{j=1}^{k} a_{jj}$$

Example 15.

Consider the following matrix:

$$\mathbf{A} = \begin{pmatrix} 153 & 143 & 133 \\ 128 & 118 & 138 \\ 128 & 161 & 108 \end{pmatrix}.$$

Use Python to find the trace of matrix $\mathbf{A}$.

Sol:

Trace of $\mathbf{A} = 379$

```
import numpy as np
A = np.array([[153,143,133],[128,118,138],[128,161,108]])
TRA = np.trace(A)
print("Trace of A =", TrA)
Output:
Trace of A = 398
```

5.1.13 Eigenvalues and Eigenvectors

Definition 15. For a $k \times k$ matrix $\mathbf{A}$, the scalars $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k$ satisfying the polynomial equation

$$|\mathbf{A} - \lambda\, \mathbf{I}| = 0$$

are called the eigenvalues (or characteristic roots) of $\mathbf{A}$.

Definition 16. Corresponding to any eigenvalue λ_i is an eigenvector (or characteristic vector) $\mathbf{u}_i \neq \mathbf{0}$ satisfying

$$\mathbf{A}\, \mathbf{u}_i = \lambda_i\, \mathbf{u}_i.$$

Example 16. Find the eigenvalue and eigenvector of

$$\mathbf{A} = \begin{bmatrix} 1.96 & 0.72 \\ 0.72 & 1.54 \end{bmatrix}$$

Sol:

```
 Python codes:
import numpy as np
from scipy import linalg
A = np.array([[1.96, .72], [.72,1.54]])
w, v = linalg.eig(A)
print("w=", w)
print("v=",v)
Output:
w=
[2.5 1. ]
v=
[[ 0.8 -0.6]
 [ 0.6  0.8]]
```

5.1.14 Covariance Matrix

The sample **covariance**

$$s_{ik} = \frac{1}{n-1}\sum_{j=1}^{n}(x_{ji} - \bar{x}_i)(x_{jk} - \bar{x}_k)$$
$$i = 1, 2, , \ldots, p, k = 1, 2, \ldots, p$$

measures the association between the i^{th} and the k^{th} variables.

Notes:

- When $i = k$, the covariance reduces to sample variance.
- $s_{ik} = s_{ki}$ for all i and k.

The covariances computed from n measurements on p variables can be organized into arrays.

Covariance: $\mathbf{S_n} = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1p} \\ s_{21} & s_{22} & \cdots & s_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ s_{p1} & s_{p2} & \cdots & s_{pp} \end{bmatrix}$

Example 17.
Consider the following data about 3 variables:

x_1	x_2	x_3
45	38	10
37	31	15
42	26	17
35	28	21
39	33	12

Derive the sample covariance matrix using the NumPy package.

Sol:

```
import numpy as np
x1 = [45, 37, 42, 35, 39]
x2 = [38, 31, 26, 28, 33]
x3 = [10, 15, 17, 21, 12]
data = np.array([x1, x2, x3])
cov_matrix = np.cov(data, bias=False)
print(cov_matrix)
Output:
[[ 15.8    9.6  -12.  ]
 [  9.6   21.7  -17.25]
 [-12.   -17.25  18.5 ]]
```

5.2 System of Linear Equations

A system of linear equations (or linear system) is a collection of one or more linear equations involving the same set of variables. For example,

$$\begin{aligned} 3x + 2y - z &= 1 \\ 2x - 2y + 4z &= -2 \\ -x + \frac{1}{2}y - z &= 0 \end{aligned}$$

is a system of three equations in the three variables x, y, z. A solution to a linear system is an assignment of values to the variables such that all the equations are simultaneously satisfied. A solution to the system above is given by $x = 1, y = -2, z = -2$ since it makes all three equations valid. The word "system" indicates that the equations are to be considered collectively, rather than individually.

A general system of m linear equations with n unknowns can be written as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m_1}x_1 + a_{m_2}x_2 + \cdots + a_{m_n}x_n &= b_m \end{aligned}$$

where $x_1, x_2, \ldots, x_n$ are the unknowns, $a_{11}, a_{12}, \ldots, a_{mn}$ are the coefficients of the system, and $b_1, b_2, \ldots, b_n$ are the constant terms.

5.2.1 Solving System of Linear Equations Using Matrix Inversion

The system of linear equations can be expressed in matrix equation of the form $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{x}$ is a column vector with n entries, and $\mathbf{b}$ is a column vector with m entries.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{12} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

If the matrix $\mathbf{A}$ is square (has m rows and $n = m$ columns) and has full rank (all m rows are independent), then the system has a unique solution given by

$$\mathbf{x} = A^{-1}\mathbf{b}$$

Example 18.

Solve the system of linear equations below using matrix inversion.

$$3x + 2y - z = 1$$
$$2x - 2y + 4z = -2$$
$$-x + \frac{1}{2}y - z = 0$$

Sol: $\mathbf{A} = \begin{bmatrix} 3 & 2 & -1 \\ 2 & -2 & 4 \\ -1 & \frac{1}{2} & -1 \end{bmatrix}$, $\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$, and $\mathbf{b} = \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix}$

$\mathbf{A}^{-1} = \begin{bmatrix} 0 & -.5 & -2 \\ 0.67 & 1.33 & 4.67 \\ 0.33 & 1.17 & 3.33 \end{bmatrix}$, $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = \begin{bmatrix} 0 & -.5 & -2 \\ 0.67 & 1.33 & 4.67 \\ 0.33 & 1.17 & 3.33 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ -2 \end{bmatrix}$

```
Python code:
import numpy as np
from scipy import linalg
A = np.array([[3, 2,-1], [2,-2,4],[-1,1./2,-1]])
b = np.array([[1],[-2],[0]])
IA = linalg.inv(A)
print("A^(-1}=",IA)
x = IA@b
print("x=A^(-1)b",x)
Output:
A^(-1}= [[ 2.22044605e-16 -5.00000000e-01 -2.00000000e+00]
 [ 6.66666667e-01  1.33333333e+00  4.66666667e+00]
 [ 3.33333333e-01  1.16666667e+00  3.33333333e+00]]
x=A^(-1)b [[ 1.]
 [-2.]
 [-2.]]
```

Example 19.

Consider the following linear system:

$$\begin{aligned} 36w + 26x + 21y + 32z &= 53 \\ 33w + 17x + 49y + 34z &= 22 \\ 32w + 34x + 17y + 33z &= 90 \\ 33w + 34x + 54y + 63z &= 71 \end{aligned}$$

(a) Write the above system in the form $\mathbf{AX} = \mathbf{b}$.

(b) Provide the Python codes to solve the linear system using matrix inversion.

(c) Suppose the unique solution to part (a) is $\mathbf{X} = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -1.0132 \\ 4.3565 \\ 0.2494 \\ -0.9071 \end{bmatrix}$, write down the Python codes to find $24w + 31x + 13y + 24z$ and calculate the corresponding value.

Sol:

(a)
```
import numpy as np
from scipy import linalg
A = np.array([[36,26,21,32],[33,17,49,34],
[32,34,17,33],[33,34,54,63]])
b = np.array([[53],[22],[90],[71]])
x = linalg.inv(A)@b
print(x)
```

(b) $\begin{bmatrix} 36 & 26 & 21 & 32 \\ 33 & 17 & 49 & 34 \\ 32 & 34 & 17 & 33 \\ 33 & 34 & 54 & 63 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 53 \\ 22 \\ 90 \\ 71 \end{bmatrix}$

(c)
```
c = np.array([24,31,13,24])
bnew = c@x
print("bnew = ",bnew)
```

$24w+31x+13y+24z = 24(-1.0132)+31(4.3565)+13(0.2494)+24(-0.9071) = 92.2$

5.2.2 Solving System of Linear Equations Using linalg's Solve Routine

The system of linear equations can be also solve using linalg's Routine in numpy package. Instead of solving $\mathbf{x} = \mathbf{A^{-1}b}$. We can obtained the solution by using the linalg's solve routine.

Example 20.

Solve the system of linear equations below using linalg's solve routine.

$$\begin{aligned} 3x + 2y - z &= 1 \\ 2x - 2y + 4z &= -2 \\ -x + \frac{1}{2}y - z &= 0 \end{aligned}$$

```
Python code:
import numpy as np
from scipy import linalg
A = np.array([[3, 2,-1], [2,-2,4],[-1,1./2,-1]])
b = np.array([[1],[-2],[0]])
x = linalg.solve(A,b)
print("x=",x)
Output:
x [[ 1.]
 [-2.]
 [-2.]]
```

5.3 Regression Analysis

5.3.1 What Is Regression?

Regression searches for relationships among variables. For example, you can observe the selling price of properties and try to understand how the area of living space on the properties, the appraised home value on the properties, and appraised land value of the properties, the selling price depend on the features, such as the area of living space on the property, the appraised home value on the property, and appraised land value of the property, and so on.

This is a regression problem where data related to each property represent one observation. The presumption is that the area of living space on the property, the appraised home value on the property, and appraised land value of the property are the independent features, while the selling price depends on them.

Generally, in regression analysis, you usually con-

sider some phenomenon of interest and have a number of observations. Each observation has two or more features. Following the assumption that (at least) one of the features depends on the others, you try to establish a relation among them. In other words, you need to find a function that maps some features or variables to others sufficiently well. The dependent features are called the **dependent variables, outputs, or responses**. The independent features are called the **independent variables, inputs, or predictors**.

Regression problems usually have one continuous and unbounded dependent variable. The inputs, however, can be continuous, discrete, or even categorical data such as gender, nationality, brand, and so on.

It is a common practice to denote the outputs with y and inputs with x. If there are two or more independent variables, they can be represented as the vector $\mathbf{x} = (x_1, \ldots, x_r)$, where r is the

number of inputs.

5.3.2 When Do You Need Regression?

Typically, you need regression to answer whether and how some phenomenon influences the other or how several variables are related. For example, you can use it to determine if and to what extent the area of living space on the property, the appraised home value on the property, and appraised land value of the property impact selling price.

Regression is also useful when you want to forecast a response using a new set of predictors. For example, you could try to predict electricity consumption of a household for the next hour given the outdoor temperature, time of day, and number of residents in that household.

Regression is used in many different fields: economy, computer science, social sciences, and so on. Its importance rises every day with the availability of large amounts of data and increased aware-

ness of the practical value of data.

5.3.3 Problem Formulation

When implementing linear regression of some dependent variable y on the set of independent variables $\mathbf{x} = (x_1, \ldots, x_r)$, where r is the number of predictors, you assume a linear relationship between y and x:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r + \epsilon.$$

This equation is the regression equation. $\beta_0, \beta_1, \ldots, \beta_r$ are the regression coefficients, and ϵ is the random error.

Linear regression calculates the estimators of the regression coefficients, denoted with $b_0, b_1, \ldots, b_r$. They define the estimated regression function $\hat{y} = b_0 + b_1 x_1 + \cdots + b_r x_r$. This function should capture the dependencies between the inputs and output sufficiently well.

The estimated or predicted response, $\hat{y}_i$, for each observation $i = 1, \ldots, n$, should be as close as

possible to the corresponding actual response y_i. The differences $y_i - \hat{y}_i$ for all observations $i = 1, \ldots, n$, are called the residuals or errors. Regression is about determining the best predicted weights, that is the weights corresponding to the smallest residuals.
To get the best weights, you usually minimize the sum of squared residuals(errors) (SSE) for all observations $i = 1, \ldots, n$, SSE $= \sum_i (y_i - \hat{y}_i)^2$. This approach is called the method of ordinary least squares(OLS).

5.3.4 Regression Performance

The variation of actual responses $y_i, i = 1, \ldots, n$ occurs partly due to the dependence on the predictors x_i. However, there is also an additional inherent variance of the output.
The coefficient of determination, denoted as R^2, tells you which amount of variation in y can be explained by the dependence on x using the particular regression model. Larger R^2 indicates a

better fit and means that the model can better explain the variation of the output with different inputs.
The value $R^2 = 1$ corresponds to $SSE = 0$, that is to the perfect fit since the values of predicted and actual responses fit completely to each other. Thus, the larger the R^2, the better fit the data to the model. However, R^2 increases as the number parameters increases. Thus we usually will use adjusted R^2 to choose the appropriate model. Another two quantity that use to make decision on model selection are Aikaike Information Criterion(AIC) and Bayesisn Information Criterion(BIC).

5.3.5 Simple Linear Regression

Simple or single-variate linear regression is the simplest case of linear regression with a single independent variable, $\mathbf{x} = x$.
When implementing simple linear regression, you typically start with a given set of input-output (x, y) pairs. These pairs are your observations.

The estimated regression function has the equation $y = \beta_0 + \beta_1 x$. Your goal is to calculate the optimal values of the predicted weights β_0 and β_1 that minimize Sum of Square of Error (SSE) and determine the estimated regression function. The value of β_0, also called the intercept, shows the point where the estimated regression line crosses the y axis. It is the value of the estimated response $y = 0$ for $x = 0$. The value of β_1 determines the slope of the estimated regression line.

5.3.6 Multiple Linear Regression

Multiple or multivariate linear regression is a case of linear regression with two or more independent variables.

If there are just two independent variables, the estimated regression function is $\hat{y} = b_0 + b_1 x_1 + b_2 x_2$. It represents a regression plane in a three-dimensional space. The goal of regression is to determine the values of the weights b_0, b_1 and b_2 such that this plane is as close as possible to the

actual responses and yield the minimal SSE.
The case of more than two independent variables is similar, but more general. The estimated regression function is $\hat{y} = \beta_0 + \beta_1 x_1 + \cdots b_r x_r$ and there are $r + 1$ weights to be determined when the number of inputs is r.

5.3.7 Polynomial Regression

You can regard polynomial regression as a generalized case of linear regression. You assume the polynomial dependence between the output and inputs and, consequently, the polynomial estimated regression function.
In other words, in addition to linear terms like $b_1 x_1$, your regression function y can include nonlinear terms such as $b_2 x_1^2$, $b_1 x_1^3$, or even $b_4 x_1 x_2$, $b_5 x_1 x_3$ and so on.
The simplest example of polynomial regression has a single independent variable, and the estimated regression function is a polynomial of degree 2: $\hat{y} = b_0 + b_1 x + \beta_2 x^2$.

Keeping this in mind, compare the previous regression function with the function $\hat{y} = b_0 + b_1 x + b_2 x^2$ used for linear regression. They look very similar and are both linear functions of the unknowns b_0, b_1, and b_2. This is why you can solve the polynomial regression problem as a linear problem with the term x^2 regarded as an input variable.

In the case of two variables and the polynomial of degree 2, the regression function has this form: $\hat{y} = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_1^2 + b_4 x_1 x_2 + b_5 x_2^2$. The procedure for solving the problem is identical to the previous case. You apply linear regression for five inputs: $x_1, x_2, x_1^2, x_1 x_2$, and x_2^2. What you get as the result of regression are the values of six weights which minimize SSE: b_0, b_1, b_2, b_3, b_4, and b_5.

5.4 Ordinary Least Squares Estimation

For the linear model with

$$\mathbf{y} = \mathbf{X}\,\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

we have

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1r} \\ X_{21} & X_{22} & \cdots & X_{2r} \\ \vdots & \vdots & & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{nr} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_r \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

and

$$\begin{aligned} y_i &= \boldsymbol{\beta}_1\,\mathbf{x}_{i1} + \boldsymbol{\beta}_2\,\mathbf{x}_{i2} + \cdots + \boldsymbol{\beta}_r\,\mathbf{x}_{ir} + \boldsymbol{\epsilon}_i \\ &= \mathbf{X}_i^T\,\boldsymbol{\beta} + \boldsymbol{\epsilon}_i \end{aligned}$$

where $\mathbf{X}_i^T = (\mathbf{x}_{i1}, \mathbf{x}_{i2}, \cdots, \mathbf{x}_{ir})$ is the i-th row of the model matrix $\mathbf{X}$.

Definition 17.
For a linear model with $\mathbf{y} = \mathbf{X}\,\boldsymbol{\beta} + \boldsymbol{\epsilon}$ any vector $\mathbf{b}$ that minimizes the sum of squared residuals

$$\begin{aligned} Q(\mathbf{b}) &= \textstyle\sum_{i=1}^{n} (y_i - \mathbf{X}_i^T\mathbf{b})^2 \\ &= (\mathbf{y} - X\mathbf{b})^T(\mathbf{y} - \mathbf{X}\mathbf{b}) \end{aligned}$$

is an ordinary least squares (OLS) estimator for $\boldsymbol{\beta}$.

For $j = 1, 2, \ldots, r$, solve

$$0 = \frac{\partial Q(\mathbf{b})}{\partial b_j} = 2 \sum_{i=1}^{n} (y_i - \mathbf{X}_i^T \mathbf{b}) \, X_{ij}$$

Dividing by 2, we have

$$0 = \sum_{i=1}^{n} (y_i - \mathbf{X}_i^T \mathbf{b}) \, X_{ij} \quad j = 1, 2, \ldots, r$$

These equations are expressed in matrix form as

$$\begin{aligned} \mathbf{0} &= \mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{b}) \\ &= \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\mathbf{b} \end{aligned}$$

or

$$\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{y}$$

These are often called the "normal" equations.

If $\mathbf{X}_{n \times r}$ has full column rank, i.e., $\text{rank}(\mathbf{X}) = k$, then

$$(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X})\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

and

$$\boxed{\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}}$$

is the unique solution to the normal equations.

If $\text{rank}(\mathbf{X}) < k$, then the solution does not exist.

Example 21. Regression Analysis:Yield of a chemical process

Yield (%)	Temperature (^oF)	Time (hr)
y	x_1	x_2
77	160	1
82	165	3
84	165	2
89	170	1
94	175	2

Multiple linear regression model

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \epsilon_i$$
$$i = 1, 2, 3, 4, 5$$

Matrix formulation:

Sol:

$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$

$$
\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} \\ \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} \\ \beta_0 + \beta_1 x_{31} + \beta_2 x_{32} \\ \beta_0 + \beta_1 x_{41} + \beta_2 x_{42} \\ \beta_0 + \beta_1 x_{51} + \beta_2 x_{52} \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \end{bmatrix}
$$

$$
= \begin{bmatrix} 1 & x_{11} & x_{21} \\ 1 & x_{12} & x_{22} \\ 1 & x_{13} & x_{23} \\ 1 & x_{14} & x_{24} \\ 1 & x_{15} & x_{25} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \end{bmatrix}
$$

$$
\begin{bmatrix} 77 \\ 82 \\ 84 \\ 89 \\ 94 \end{bmatrix} = \begin{bmatrix} 1 & 160 & 1 \\ 1 & 165 & 3 \\ 1 & 165 & 2 \\ 1 & 170 & 1 \\ 1 & 175 & 2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \end{bmatrix}
$$

Example 22. Refer to example 21, obtain the OLS estimate, $\mathbf{b}$.

$$\textit{Sol: } \mathbf{y} = \begin{bmatrix} 77 \\ 82 \\ 84 \\ 89 \\ 94 \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & 160 & 1 \\ 1 & 165 & 3 \\ 1 & 165 & 2 \\ 1 & 170 & 1 \\ 1 & 175 & 2 \end{bmatrix}, \text{ then}$$

$$\mathbf{X^T X} = \begin{bmatrix} 5 & 835 & 9 \\ 835 & 139,575 & 1,505 \\ 9 & 1,505 & 19 \end{bmatrix}, \mathbf{X^T y} = \begin{bmatrix} 426 \\ 71,290 \\ 768 \end{bmatrix}$$

$$\mathbf{X^T X^{-1}} = \begin{bmatrix} 2.149 & -1.289 & 0.278 \\ -1.29 & 0.0078 & -0.0056 \\ 0.278 & -0.0056 & 0.361 \end{bmatrix} \mathbf{X^T y} = \begin{bmatrix} 426 \\ 71,290 \\ 768 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} = \mathbf{X^T X^{-1} X^T y} = \begin{bmatrix} 2.149 & -1.289 & 0.278 \\ -1.29 & 0.0078 & -0.0056 \\ 0.278 & -0.0056 & 0.361 \end{bmatrix} \begin{bmatrix} 426 \\ 71,290 \\ 768 \end{bmatrix} = \begin{bmatrix} -105.22 \\ 1.144 \\ -0.389 \end{bmatrix}$$

Python code:

```
import numpy as np
from scipy import linalg
y = [77,82,84,89,94]
x0 = np.ones(5).reshape(-1,1)
x1 = np.array([160,165,165,170,175]).reshape(-1,1)
x2 = np.array([1,3,2,1,2]).reshape(-1,1)
x = np.hstack((x0, x1,x2))
xtx = x.T@x
xty = x.T@y
b = linalg.inv(xtx)@xty
print("b=",b)
```

Output:

```
b= [-105.22222222    1.14444444   -0.38888889]
```

Example 23.

Consider the data below:

y	x_1	x_2	x_3	x_4
80	90	112	3	301
45	65	92	0	425
49	95	82	2	285
31	81	95	0	412
25	65	135	5	420
58	73	125	3	415
54	98	175	7	512
24	62	132	5	175
40	94	105	6	310
36	91	115	1	330

Determine the predicted value for the mean score of y with $x_1 = 78$, $X_2 = 145$, $x_3 = 4$, and $x_4 = 487$.

Sol:

$\hat{Y} = -9.96 + 0.5800(78) + 0.0500(145) + -0.9600(4) + 0.0100(487) = 44.56$

```
import numpy as np
from scipy import linalg
y = [80, 45, 49, 31, 25, 58, 54, 24, 40,36]
n = len(y)
x0 = np.ones(n).reshape(-1,1)
x1 = np.array([90,65,95,81,65,73,98,62,94,91]).reshape(-1,1)
x2 = np.array([112,92,82,95,135,125,175,132,105,115]).reshape(-1,1)
x3 = np.array([3,0,2,0,5,3,7,5,6,1]).reshape(-1,1)
x4 = np.array([301,425,285,412,420,415,512,175,310,330]).reshape(-1,1)
```

```
x = np.hstack((x0, x1,x2,x3,x4))
x, y = np.array(x), np.array(y)
xtx = x.T@x
xty = x.T@y
beta = linalg.inv(xtx)@xty
xnew = np.array([1,78, 145, 4, 487])
y_new = np.inner(xnew, beta)
print("beta hat=", beta)
print("y hat=", ynew)
Output:
beta hat= [-9.96  0.58  0.05 -0.96  0.01]
y hat= 44.56
```

Example 24.

A researcher believes that the number of days the ozone levels exceeded 0.2ppm (y) depends on the seasonal meteorological index (x). The following table gives the data.

Index	16.3	17.1	17.6	17.1	16.6	18.1	14.0	16.5	17.8	16.8
Days	86	107	107	114	88	91	64	74	77	65

You fit the above data to $y = \beta_0 + \beta_1 x + \epsilon$, where Y is the number of days the ozone levels exceeded 0.2ppm, and X is the seasonal meteorological index.

(a) Write the model above in the form $\mathbf{y} = \mathbf{x}\boldsymbol{\beta} + \boldsymbol{\epsilon}$

(b) Write the Python commands and output using matrix formulation to obtain the estimate of β_0 and β_1.

(c) You are given that $R^2 = \frac{SSR}{SST}$ and adjusted R^2, $R^2_{Adj} = 1 - \frac{SSE/(n-p)}{SST/(n-1)}$, where

- n is the number of observations.
- p is the number of parameters in the model.
- $SSR = \boldsymbol{b}^T\mathbf{X}^\mathbf{T}\mathbf{y} - \frac{1}{n}\mathbf{y}^\mathbf{T}\mathbf{J}\mathbf{y}$, where $\mathbf{J}$ is an $n \times n$ matrix of one.
- $SSE = \mathbf{y}^\mathbf{T}\mathbf{y} - \boldsymbol{b}^T\mathbf{X}^\mathbf{T}\mathbf{y}$.
- $SST = \mathbf{y}^\mathbf{T}\mathbf{y} - \frac{1}{n}\mathbf{y}^\mathbf{T}\mathbf{J}\mathbf{y}$.

Write the Python commands and output to calculated R^2 and adjusted R^2.

Sol:

(a)
$$\begin{bmatrix} 86 \\ 107 \\ 107 \\ 114 \\ 88 \\ 91 \\ 64 \\ 74 \\ 77 \\ 65 \end{bmatrix} = \begin{bmatrix} 1 & 16.3 \\ 1 & 17.1 \\ 1 & 17.6 \\ 1 & 17.1 \\ 1 & 16.6 \\ 1 & 18.1 \\ 1 & 14.0 \\ 1 & 16.5 \\ 1 & 17.8 \\ 1 & 16.8 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} \begin{bmatrix} +\epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \\ \epsilon_8 \\ \epsilon_9 \\ \epsilon_{10} \end{bmatrix}$$

(b)
```
import numpy as np
y = np.array([86,107,107,114,88,91,64,74,77,65])
x = np.array([[1,16.3],[1,17.1],[1,17.6],[1,17.1],[1,16.6],
              [1,18.1],[1,14.0],[1,16.5],[1,17.8],[1,16.8]])
xtx = X.T@x
xty = X.T@y
b = np.linalg.inv(xtx)@xty
print(b)
Output:
[-47.160286469443236 8.008355358513427]
```

(b)
```
import numpy as np
y = np.array([86,107,107,114,88,91,64,74,77,65])
x = np.array([[1,16.3],[1,17.1],[1,17.6],[1,17.1],[1,16.6],
              [1,18.1],[1,14.0],[1,16.5],[1,17.8],[1,16.8]])
n = len(y)
p = 2
xtx = X.T@x
xty = X.T@y
b = np.linalg.inv(xtx)@xty
print("beta hat =", b)
yty = y.T@y
bxty = b.T@xty
J = np.ones((n,n))
```

```
ytJy = y.T@J@y
SSR = bxty-ytJy/n
SSE = yty-bxty
SST = yty-ytJy/n
print("SSE=", SSE)
print("SSR=", SSR)
print("SST=",SST)
RSq = SSR/SST
AdjRSq = 1-(SSE/(n-p))/(SST/(n-1))
print("Rsq=",RSq)
print("AdjRsq=", AdjRSq)
Output:
beta hat = [-47.160286469443236 8.008355358513427]
SSE= 2075.8751811772527
SSR= 752.2248188227532
SST= 2828.100000000006
RSq= 0.2659823976601788
AdjRSq= 0.17423019736770107
```

Example 25.

A local Bank wants to estimate the number of foreclosures per 1000 houses sold. Bank officials think that the number of foreclosures is related to the size of the down payment made by house buyers. The following table contains foreclosure and down payment data.

Down Payment Size (% of purchase price)	Number of Foreclosures per 1000 houses
10	40
20	9
14	29
12	27
18	15
16	23

You fit the above data to $y = \beta_0 + \beta_1 y + \epsilon$, where y is the number of foreclosures per 1000 houses sold, and x is the size of the down payment made by house buyers. You are given:

- n is the number of observations.
- p is the number of parameters in the model.
- $SSR = \boldsymbol{b}^T\mathbf{x}^\mathbf{T}\mathbf{y} - \frac{1}{n}\mathbf{y}^\mathbf{T}\mathbf{J}\mathbf{y}$, where $\mathbf{J}$ is an $n \times n$ matrix of one.

Write down the Python commands to calculated SSR.

Sol:

```
import numpy as np
from scipy import linalg
y = np.array([40,9,29,27,15,23])
n = len(y)
x = np.array([[1,10],[1,20],[1,14],[1,12],[1,18],[1,16]])
p = 2
xtx = x.T@x
xty = x.T@y
b = linalg.inv(xtx)@xty
print("beta hat =", b)
yty = y.T@y
bxty = b.T@xty
J = np.ones((n,n))
ytJy = y.T@J@y
SSR = bxty-ytJy/n
print("SSR=", SSR)

Output:
SSR= 554.4142857142783
```

Example 26.

A local Bank wants to estimate the number of foreclosures per 1000 houses sold. Bank officials think that the number of foreclosures is related to the size of the down payment made by house buyers. The following table contains foreclosure and down payment data.

Down Payment Size (% of purchase price)	Number of Foreclosures per 1000 houses
10	40
20	10
14	28
12	25
18	14
16	18

You fit the above data to $y = \beta_0 + \beta_1 y + \epsilon$, where y is the number of foreclosures per 1000 houses sold, and x is the size of the down payment made by house buyers. You are given:

- n is the number of observations.
- p is the number of parameters in the model.
- $SSE = \mathbf{y^T y} - \boldsymbol{b}^T \mathbf{x^T y}$.

Write the Python commands to calculated SSE.

Sol:

```
import numpy as np
from scipy import linalg
y = np.array([40,10,28,25,14,18])
x = np.array([[1,10],[1,20],[1,14],[1,12],[1,18],[1,16]])
n = len(y)
p = 2
xtx = x.T@x
xty = x.T@y
b = linalg.inv(xtx)@xty
print("beta hat =", b)
yty = y.T@y
bxty = b.T@xty
SSE = yty-bxty
print("SSE=", SSE)
Output:
SSE= 59.37142857143772
```

Example 27.

Consider the data shown below:

y	x	y	x
20	16	19	13
18	10	19	11
19	11	20	16
18	9	17	6
20	14	19	11
17	6	19	11
21	17	20	15
18	10	18	9

You fit the above data to $y = \beta_0 + \beta_1 x + \epsilon$. You are given that:

- n is the number of observations.
- p is the number of parameters in the model.
- $\boldsymbol{b} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$.
- $SSE = \mathbf{y^T y} - \boldsymbol{b}^T \mathbf{X^T y}$.
- $MSE = \frac{SSE}{n-p}$
- $SE(b_j) = \sqrt{MSE \times C_{jj}}$, where C_{jj} is the diagonal element of the $(\mathbf{X^T X})^{-1}$ corresponding to b_j.

Compute $SE(b_1)$.

Sol:
$SE(b_1) = 0.02258622945422641 6$

```
from scipy import sqrt,linalg
import numpy as np
y = np.array([20,18,19,18,20,17,21,18,
19,19,20,17,19,19,20,18])
x = np.array([[1,16],[1,10],[1,11],[1,9],[1,14],[1,6],[1,17],[1,10],
[1,13],[1,11],[1,16],[1,6],[1,11],[1,11],[1,15],[1,9]])
xtx = x.T@x
xty = x.T@y
b = linalg.inv(xtx)@xty
n = len(y)
p = 2
print("beta hat =", b)
yty = y.T@y
bxty = b.T@xty
SSE = yty-bxty
MSE = SSE/(n-p)
Cjj = linalg.inv(xtx)[1,1]
SEbeta = sqrt(MSE*Cjj)
print("SEbeta=", SEbeta)
Output:
SEbeta= 0.0204300132350792
```